NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF GRAPHICAL LANGUAGE FOR ACCESSING DATABASE

by

Alparslan Horasan

June 1986

Thesis Advisor: C. T. Wu

Approved for public release; distribution is unlimited.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*) UNCLASSIFIED
Implementation of Graphical Language for Accessing Database

12 PERSONAL AUTHOR(S)
Alparslan Horasan

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Masters Thesis	FROM _____ TO _____	1986 June 20	81

16. SUPPLEMENTARY NOTATION

17 COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	GLAD, Definition Window, Schema Design Area, Manipulation Window, Help Windows

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)
This thesis is a part of the implementation of a new graphics user interface for accessing a database proposed in paper (WU86). As a result of this study, the data definition language of the proposed graphics user interface GLAD (Graphical Language for Accessing Database) has been implemented. This interface allows a user to create a database schema graphically. It is easy to learn and easy to use, in spite of conventional query languages. This thesis first discusses the general concepts of database and introduces the system that the implementation was achieved, then reviews the conventional query language and previously proposed graphical user interfaces. After describing the major features of GLAD, the implementation is explained in detail. A listing of the program that achieves the interface is also provided.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. C. T. Wu	22b. TELEPHONE (*Include Area Code*) 408-646-3391 22c. OFFICE SYMBOL 52Wq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Implementation Of Graphical Language For Accessing Database

by

Alparslan Horasan
First Lieutenant, Turkish Air Force
B. S., Turkish Air War Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1986

ABSTRACT

This thesis is a part of the implementation of a new graphics user interface for accessing a database proposed in paper [WU86]. As a result of this study. the data definition language of the proposed graphics user interface GLAD(Graphical Language for Accessing Database) has been implemented. This interface allows a user to create a database schema graphically. It is easy to learn and easy to use. in spite of conventional query languages. This thesis first discusses the general concepts of database and introduces the system that the implementation was achieved. then reviews the conventional query languages and previously proposed graphical user interfaces. After describing the major features of GLAD. the implementation is explained in detail. A listing of the program that achieves the interface is also provided.

TABLE OF CONTENTS

I. INTRODUCTION

A database model is a vocabulary for describing the structure and processing of a database. DDL(data definition language) is the vocabulary for defining the structure of the database. DDL specifies the conceptual scheme of the database. The data definition language is used when the database is designed, and when the design is modified. DML(data manipulation language) is the vocabulary for describing the processing of the database. Processing could be changing or retrieving the database data. DDL and DML compose a query language.

A query language can be efficiently used only by sophisticated users such as database administrators and system designers. Experiences in using these languages have shown even those people with computer science background often have difficulty using these languages. However, nowadays. they are not the only ones who are dealing with databases. Casual users who are not computer science professionals such as accountants, clerks. statisticians are the most frequent users of the databases. They may not have the patience, ability, or desire to learn and to use these query languages. Difficulties in using these query languages will be reviewed in Chapter 2.

Because of the diversity of the users who are dealing with the databases from casual to sophisticated users and difficulties using the query languages, a better user interface is needed for the databases.

In the paper [WU86], a good user interface is described as capable of supporting computer science professionals and the growing community of users who have to access the information in the database management system but who are not trained in the use of such systems. To support those users, a good user interface must have some characteristics. This paper categorizes them into four categories. The first one is ease of learning. A good user interface must be easy to learn because of the diversity of the users. Second is ease of using because of the same reason above. The third category is that it must be able to show what data is stored in the database and what relations exist among them which is known as *database schema*. Over the past years, database schemas have been used to to explain the conceptual scheme of the database. However, it has never been used for interfacing the database so the user does not have to remember the types of records and attributes and relations. The fourth one is the power of the interface to be able to handle complex queries.

Graphical user interfaces have been proposed by so many researchers as a solution to better user interface. The upsurge in development of graphical user interfaces stems from the capability of the graphics in terms of visualizing the concepts. Graphics has been used in many ways. However, they all lacked fulfilling the features of a better user interface which is explained in [WU86]. Prior researches in this field will be reviewed in Chapter 3.

One of the approaches of graphical user interfaces is GLAD(Graphical Language in Accessing Databases). GLAD attempts to eliminate the lacking

features of a good user interface that exist in other approaches. GLAD's approach is to be able to use to database schema to access the database. The features of GLAD will be reviewed in Chapter 4.

GLAD consists of two components which they are the same as the other database management systems: DDL and DML. This thesis mainly covers the implementation part of GLAD's DDL. DDL was implemented on system *isiv* which has a graphics capability that enabled us to implement the DDL part of GLAD. Implementation will be reviewed in chapter5.

Graphics workstation that DDL was implemented has a high resolution graphics display with an addressable screen space of 1280 pixels(height) by 2024 pixels(width) in a 19-inch display. Workstation has a keyboard and mouse as a standard input. Mouse consists of three buttons: left most butto used for selecting and moving windows, middle button used for pop-up menus and right most button used for creating and erasing the windows. Actions are done either by these three buttons or selecting the action from the pop-up menu and letting the *go* command run.

This workstation offers multiple window capabilities and a desktop methaphor. Multiple windows enable the user to deal with different windows without loosing his attention.

Desktop consists of icons, each icon representing a program or a separate device. For instance, Unix operating system is represented with Cshell icon.

Once the user creates the Unix OS window by selecting and letting the Cshell icon run. he gets a separate device that works as a separate unix workstation.

With this graphics capability, user can create a database schema by creating the records and the relations. The goal of this thesis is to implement the graphical DDL facilities, so user can see which data is stored in the database and which relations exist among them by scanning the schema.

II. DIFFICULTIES IN USING THE QUERY LANGUAGES

In this chapter we review the major difficulties in using the query languages. Following factors can be considered as the major reasons for the difficulty in using and understanding the query languages.

There are so many things to be recalled by the user. Before the user expresses a query, he has to remember the names of the record types and the attributes. User can only find the details of the attributes such as the format and the units of the attributes by exploring the data and looking the attribute definitions up in the dictionary. To determine the meaning of acronyms used to represent record types and their attributes is another problem. These problems become worse when the database has hundreds of records and thousands of attributes.

Most query languages are based on mathematical concepts such as predicate calculus or algebra or set theory. These mathematical concepts leads to a language with a solid foundation; often, however, users don't relate to mathematical concepts such as range variables, join clauses or projections. More explicit models should be used to support the non-expert user interface to complex data.

To formulate a complex query correctly on the first try is considered as success in dealing with the database. There is always a doubt as to whether the query is complete or whether some conditions are missing. Sometimes user does

not have the complete query in his mind. Instead of a complete query, building a query in a piecemeal fashion with feedback of partial results would be more beneficial to the user. These query languages does not support experimental, exploratory nature of formulating queries in piecemeal fashion.

Complex schemas with hundreds of elements in it, could be overwhelming to deal with it. Most query systems have only two levels of detail at the schema level: record(relation set) and attribute level. Even when second level is suppressed, the user still has to locate the relevant record types. At the attribute level, there are thousands of attributes of a record type to be checked to formulate a query. It is very hard to find the relevant record types and attributes to formulate a query.

Complex databases often have a large number of data sets that deal with different subject matters. It is a hard job to know what and where data is being kept. Lack of viewing the database generally with the query systems, blocks the users to select subject matters that are of interest.

III. PRIOR RESEARCHES

In this chapter, we review some of the previously proposed graphics user interfaces to understand the concept of graphical user interfaces.

SDMS (Spatial Data Management System) is a sophisticated data browser. The representation of the query and the output data is done thorough the use of "icons", which are graphical tokens representing database objects. With this system the entire database is shown to the user as icons on multiple graphics terminals. A user could move his cursor to an icon of interest and zoom to find more detail.

The graphical presentation of the information encourages browsing and requires less prior knowledge of the contents and organization of the database. That means, user does not have to specify the information precisely and does not need to know exactly where in the DBMS the information is stored.

SDMS uses three color raster scan displays and one mouse for presenting and accessing the data. One of the screens presents the entire icons, representing different types of data in the database. (This feature corresponds to the use of icon in the system *isiv*. This system also uses icons not only to represent the database but also to represent different types of applications or systems programs). The other screen displays the magnified portion of this data. By

using the cursor the entire data can be scanned. By zooming the Icons, the information on this specific icon can be retrieved.

The weakness of this system is that it does not support complicated query languages. For example, if a user wants to compare the informations of two icons, he can't submit a query to do this. He has to find these two icons, get the information from them and compare them. This shows, SDMS only provides weak query languages.

TIMBER (Text, Icon and Map Browser for Extended Relations) is a user friendly, graphics oriented browser for relational database. Timber is more sophisticated, comparing with SDMS. In addition to data browsing. TIMBER also gives a capability to browse texts, maps, and icons.

This system provides a sophisticated visual interaction with the data in a text file. The text stored as database objects can be browsed and updated by users with this system. It also provides a capability to display the geographic data.

Mainly, TIMBER is a sophisticated version of SDMS. But it does not provide a means of representing the data and the relation between them, which is called as database schema.

One of the systems which uses graphics devices as tools to interface to databases is GUIDE (Graphical User Interface for Database Exploration). This system contains subject directories, help messages, zooming facilities to the relevant part of the database schema, and partial query formulation with intermediate results.

This system offers a graphics interface to the user. The database schema is displayed as a network of entity and relationship types.

During the data definition stage, Data base Administrator (DBA) provides information about the schema, in addition to information on entities (objects), relationships, and their attributes, their examples and explanations of these objects. The graphical layout of the schema is fed in to the system in this stage.

In terms of visualizing database concepts, this system provides a better interface then those mentioned above. But lack of aggregate functions, use of two screens (one as a key board and also a query result presentation, and another for schema representation) and use of two separate diagrams (Entity/Relationship diagram and subject directory) for representing database schema are considered to be weak points of GUIDE.

IV. DESCRIPTION OF GLAD

This chapter describes the features of GLAD.

The main objective of GLAD is to provide users with fast, easy access to large volumes of data. GLAD achieves this objective by displaying a diagram of the database schema, representing the data stored in the database and the relations among them. This diagram is capable of representing real world abstraction concepts such as aggregation, generalization, and classification.

Aggregation is a grouping of objects and subobjects. An aggregate object consists of atomic and non-atomic objects. Atomic objects are the ones that they are an aggregation of one system-defined or a user defined "base" objects(string, number, boolean, subrange, enumeration). An aggregate object is represented with a rectangle with the name of the object written in it. The non-atomic subobject of an aggregate object is represented with a separate rectangle, with the name of the object written in it. The relation between them is represented with a solid line.

Generalization is a grouping of objects which they can be regarded as a member of a general category. The objects that comprise the generalized object are called specialized objects. Generalized object is an abstract representation of a group of objects. Each member of a generalized object can also be an abstract representation of an another group of objects. This level of abstraction provides

user to deal with the same type of data without loosing his attention in the diagram. A generalized object is represented with a nested rectangle and the name of the object written in it.

An association between a generalized object and another object is represented with a solid line as it is represented with aggregate objects. When one or both of the objects are specialized objects the relation between them is represented with dashed lines.

The last abstraction concept we mention here is Classification. Classification imposes that each data item stored in the database is an information about some object. Each data item of an objet is called as the member of the object.

To indicate that an object can have either one subobject or another, a circle is used attached to the upper line of the rectangle. This circle can be used by all the objects in the diagram. This type of relation among the objects is called as disjunctive relation.

GLAD diagram to represent the database schema fulfills one of the features of a good user interface: descriptiveness. The other features, easiness in learning and using, of a good user interface is also fulfilled effectively by GLAD. GLAD is easy to learn and to use. User does not have too much to learn to create the diagram. Only concepts that user has to learn is that the regular and nested rectangles, solid and dashed rectangles and circle. By knowing the meanings of the components of a diagram and the use of the system, user is ready to create the

diagram. As to using the system, interaction is done with the mouse, by moving the mouse to the desired operations and clicking the buttons.

How the objects and the relations among them are represented will be reviewed in Chapter 5. Chapter 5 also talks about the use of the system in order to create the diagram to define the database. One of the features of a good user interface, powerfulness in expressing the complex queries, is mainly the concern of manipulation part. Since the concern of this thesis is to be able to define a database by using graphics facilities, manipulation of the database is not detailed.

V. IMPLEMENTATION

A. TOP LEVEL INTERFACE

Graphical database management system is represented as an icon on the desktop. As the other icons represent different types of application or system programs, database icon represents the windows opened for graphical database management system. A window can be created either by putting the mouse on top of the icon and clicking the right most button, or selecting the option from the pop-up menu by middle button and letting the "go" command run.

When the DB window is created, a top level menu presents the top level options for the database. Top level menu consists of 4 option boxes, representing four types of actions to be performed: *Define DB, Manipulate DB, Help, Quit. Define DB* box represents the window in which the creation of the database schema is performed. While creating the database schema, the records are also created, and the relations between the records are set. Creation of the database schema and setting the relations between them will be detailed later in this chapter.

Even the concern of this chapter is *Define DB* window. I would like to briefly mention of each top level option. One of the windows opened for the manipulation of the database which will not be detailed further, is *Manipulate DB* window. The actions such as update, query, retrieve are achieved on this window.

To keep the completeness of idea of GLAD *Manipulate DB* window is represented with a blank window. When *Manipulate DB* window is created a menu asking the name of the database to be dealt with is presented. Designing the actions on manipulating the database is not the concern of this thesis.

Help option box, on the top level menu, represents the windows that give information about defining and manipulating the database. When the *Help* option box. on the top level menu, is selected with the mouse and right most button is clicked. the top level menu goes away and another menu containing the options *DB Definition* and *DB Manipulation* is presented. This gives the user the opportunity to learn how DB is created and manipulated. If the user wants information about defining a database, he puts the mouse in the *DB Definition* option box and clicks the right most button. This creates the window that will be filled out with the information about defining a database. Pop_up menu with an *Exit* option is always available to exit the window and go back to *Help* menu. The other window *DB Manipulation* will also be filled out with information about manipulating the database. There is also *QUIT* option to go back to top level menu.

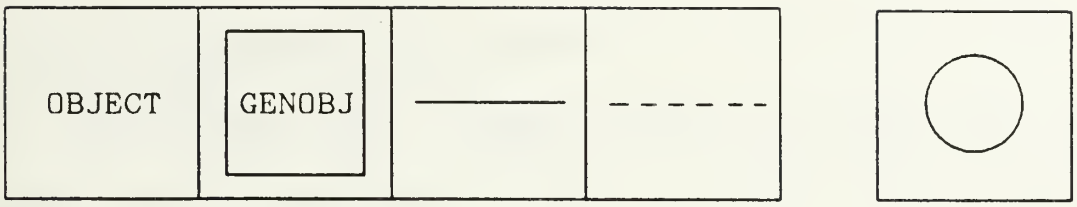
The windows .describing the actions. can be put to the corners of the screen so they can be read in case information is needed. However, for a sophisticated user. there might be no need to review. Therefore, these windows might be reviewed or not depending on the knowledge level of the user on dealing with the database.

The last option box on the top level menu is *QUIT*. When this box is selected and the right most button is clicked, all the windows associated with the graphical database management system are killed. Killing a window also kills all the windows associated with the program. After killing the window, user either gets to desktop or to the other windows created before the killed window.

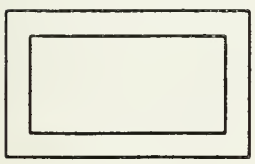
B. HOW TO CREATE THE DEFINITION WINDOW

Once the user decides to define a database, he puts the mouse in *Define DB* option box, on the top level menu, and clicks the right most button. This creates the *Define DB* window, which is going to be used to draw the diagram representing the database schema. When the user gets the window, he is ready to start creating the database schema. Only mouse is used to do this. Window consists of three sections : the section that the type options are presented which is on the upper left area of the window, the section that mode options are presented which is in the upper right area of the window and the section that the diagram is drawn, which I call *schema design area*. Type options presents the types of the objects and relations to be created. Mode options presents the actions to create and edit them.

A diagram, representing the concepts mentioned in Chapter 4, consists of five components. The components are presented on the upper left corner of the window, which I call the *type option boxes*. The components of the diagram and what they represent is shown in Figure1.



Represents an aggregate object or a non-atomic object



Represents a Generalized object

———— Represents an association between objects

- - - - - - - - - Represents an association between objects when one or both associated objects are specialized objects

Used by aggregate and generalized objects. Represents that an object can belong either one object or another

Figure 1

When *Define DB* window is first opened, a menu asks the user to enter the name of the database to be defined. Diagram is saved under this name. This diagram can be also called by the manipulation part with this name. Diagrams also could be represented with icons. By giving a meaningful shape, each icon can represent a diagram without giving a specific name to it. Benefit of using icons instead of names is not to have to remember the names of the databases.

The *mode* options, on the upper right area of the window, gives the user the capability to draw and edit the diagram. Modes can be selected by clicking the left most button. Each time a mode is selected, the box that represents the mode is highlighted. Mode options consists of four components: *Draw*, *Move*, and *Delete*. *Draw* mode is used for creating the aggregate and generalized objects and creating the links between the objects. This mode is also used to draw the names of the objects. *Move* mode is used to move the object in the schema design area. By moving the objects, they can be put to the appropriate places in the diagram. *Delete* mode is used for deleting an object. The mode option boxes is shown in Figure 2.

Figure 2

C. HOW TO DRAW THE DIAGRAM

When *Define DB* window is presented the default mode is *Draw* mode. because there is nothing created on the schema design area. User has a choice to begin with either aggregate objects or generalized objects. If he chooses to create an aggregate object. he selects the type option *Object* and clicks the left most button. This brings a menu that asks the name of the object and the relations existing with other objects. The menu is sent by pushing the middle button and letting the *go* command run.

After this menu is sent. another menu is presented that asks the attributes of the object. After the attributes of the object is received. user is ready to create his object by drawing the rectangle. To do this. user puts the mouse in the schema design area and pushes the left most button which will be the upper left corner of the rectangle. With the movement of the mouse a rectangle is created as being the last position of the mouse is the lower right corner of the rectangle. When the mouse is released a rectangle created and the name of the object is drawn in it.

To represent that the objects have a disjunctive relation. user has to touch the box. in which there is a circle. before drawing his rectangle. If circle box has been touched. the rectangle comes with a circle in it.

In order to create a generalized object. user puts the mouse in type option *GENOBJ* box and clicks the left most button. This prompts a menu asking the names of the objects. comprising the generalized object and the name of the

objects which the object has a relation. Also the circle box can be touched to put a circle inside the box to represent a disjunctive relation.

After all the objects in the schema have been created, they can be put anywhere in the *schema design area*, by selecting the *Move* mode. An object can be moved by selecting the object with mouse and pushing the left most button. By moving the mouse, object is also moved to the place which is appropriate in the schema. By releasing the button, object is set to the new position in the diagram. All the objects can be put to the appropriate places in the schema with this method.

Before creating the relations, if there is, the objects with wrong names, or unwanted object can be deleted in *Delete* mode. To delete an object, user puts the mouse in the rectangle, representing the object to be deleted and clicks the left most button. This erases the rectangle(object).

After the objects have been put to the appropriate positions in the diagram, we can go back to *Draw* mode and create the links representing the relations between objects.

To draw a solid line between rectangles(objects), which represents and association between them, line box is selected with left most button. Now, user is ready to draw the lines and link the associated objects. There is only one action required to draw the lines. It is to put the mouse in the drawing area and click the left most button. This creates the links with respect to the relations that

objects have. (Relations of the objects have been entered before. while creating the objects.)

The dashed lines. which represents an association between objects when one or both associated objects are specialized objects, can be drawn with the same method above. Since there is no library function to draw dashed lines, it is also represented with solid lines. This can be later modified with dashed lines.

Define DB window also provides a pop-up menu. Pop-up menu consists of two components: *Clear* and *Exit*. *Clear* command erases everything in the *schema design area*. *Exit* command is used to exit the window by saving the schema

drawn in the schema design area. Before exiting the window a menu is presented reminding the user that he diagram has been saved with the name which was first put when entering the window. This menu also asks whether user wants to continue his actions to add more objects or edit the objects. If user wants to go back to the schema, he can get his diagram back by clicking the *yes* option in the menu. With *no* option he can go back to the top level menu by exiting the *Define DB* window.

D. IMPLEMENTATION

First of all I would like to explain what modules do, what kind of actions they correspond in the process of using the system and what kind of relation exist between them. I would like to also explain the library functions in sequence of their appearance in the modules, correspondingly in the entire program.

As it is shown in Figure3. main program calls four functions. Namely. *Defwindow*. *Manipwindow*. *Help*. *Quit*. Function *Defwindow* handles the opening of the *Define DB* window and the creation of the database schema on this window. *Manipwindow* only opens a blank window to represent the window *Manipulate DB*. Function *Help* first introduces the help menu and then creates the windows. that explain the required actions to deal with this graphical database management system. *Quit* is the function that erases all the windows associated with the program.

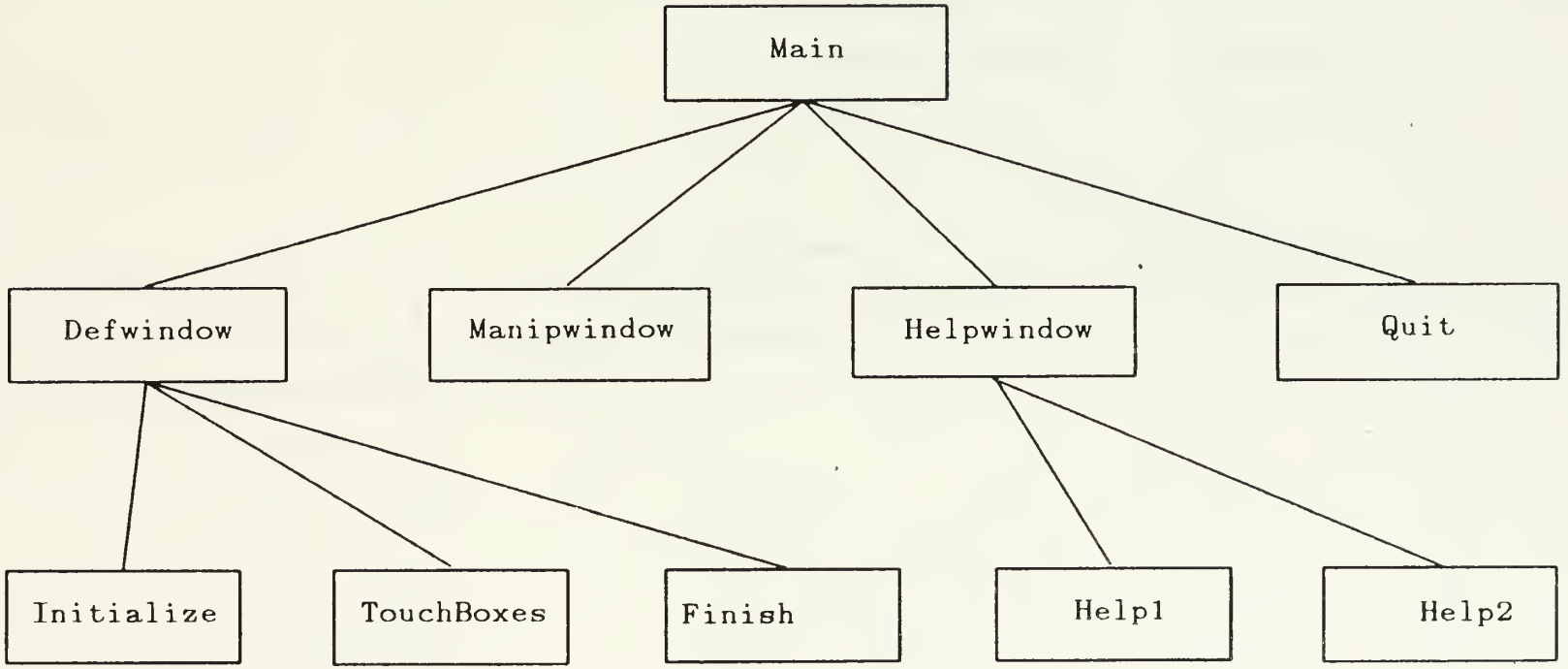

Main
Defwindow
Manipwindow
Helpwindow
Quit
Initialize
TouchBoxes
Finish
Help1
Help2

Figure 3.a

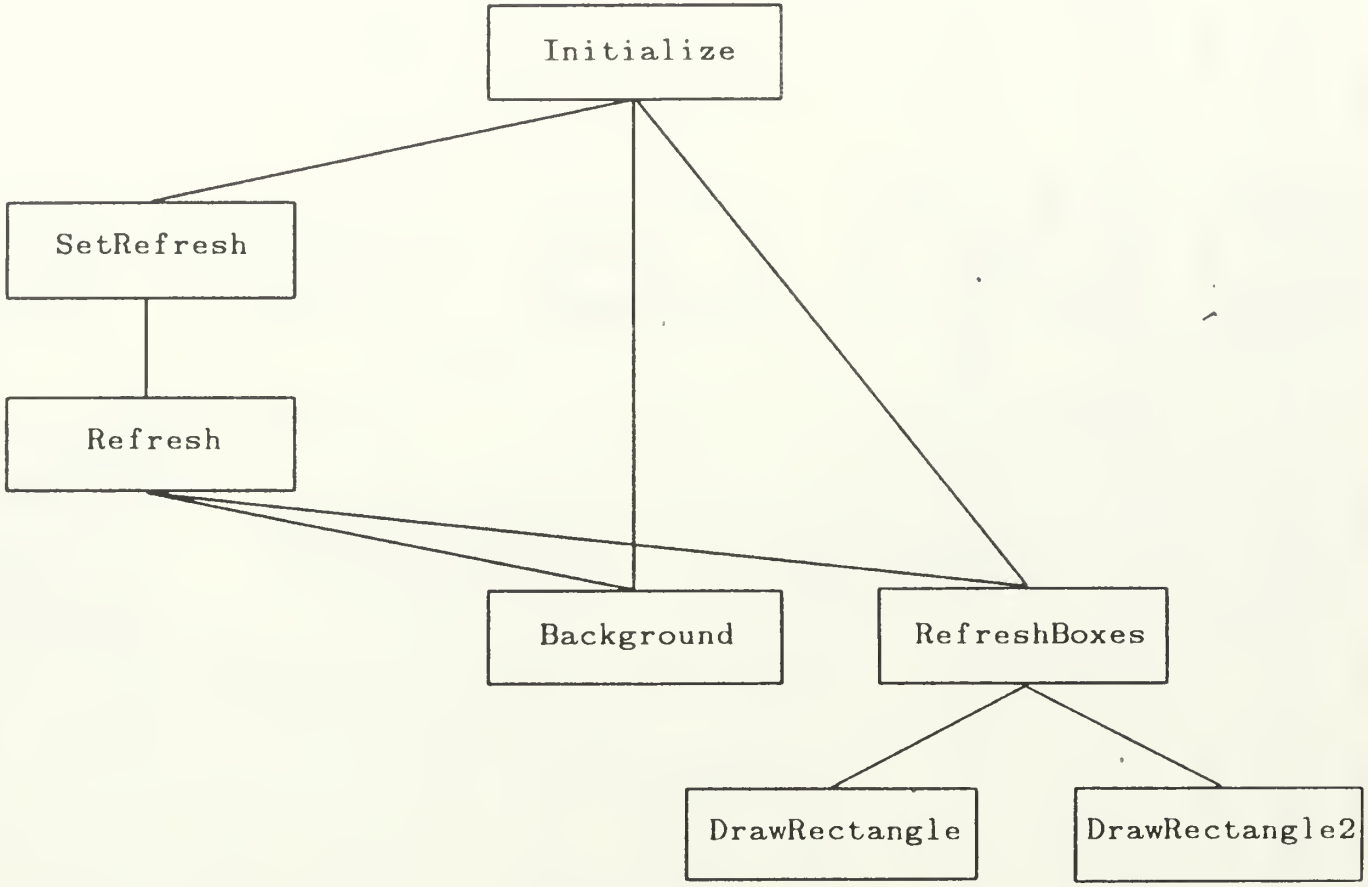

Initialize
SetRefresh
Refresh
Background
RefreshBoxes
DrawRectangle
DrawRectangle2

Figure 3.b

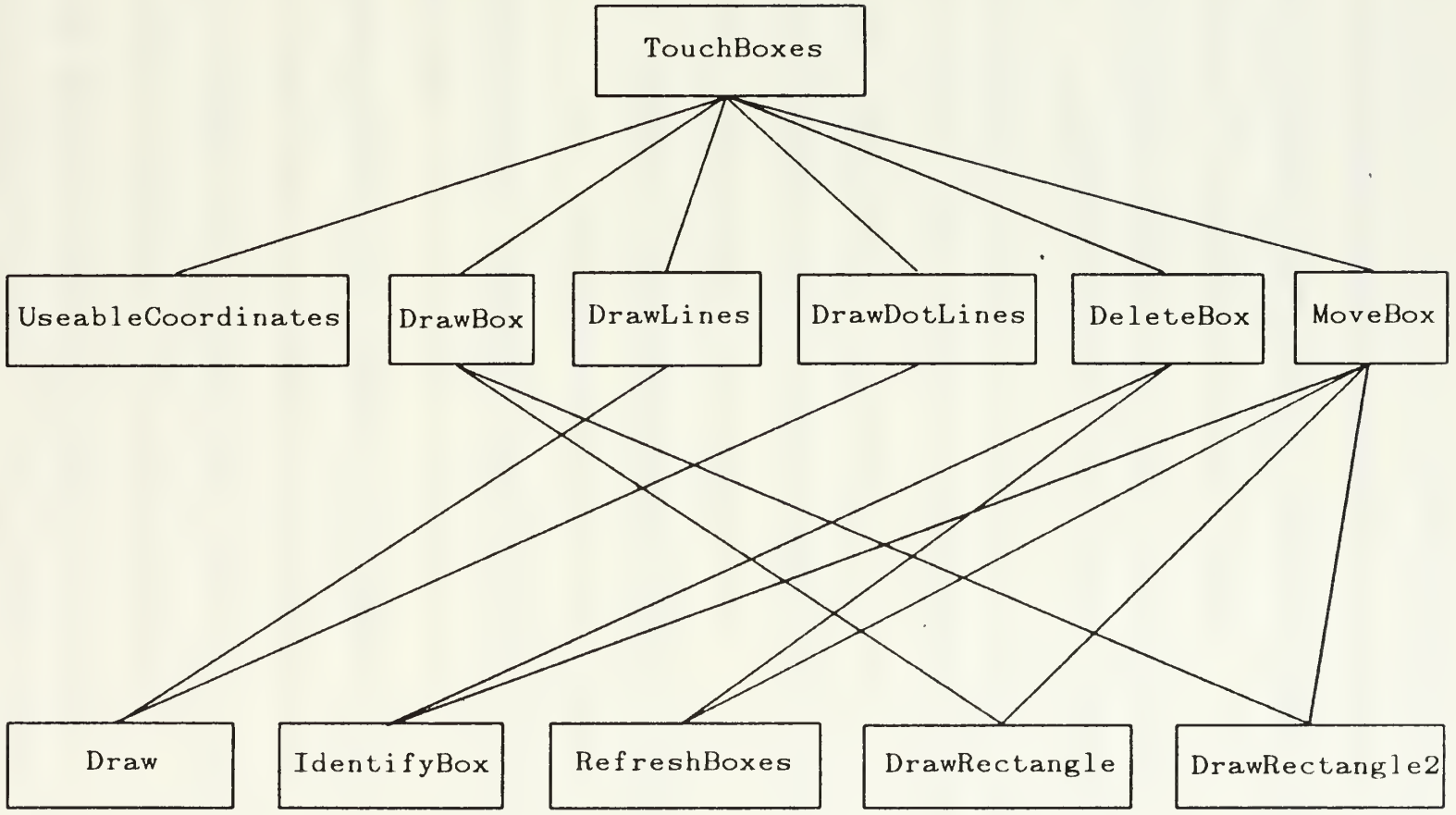

TouchBoxes
UseableCoordinates
DrawBox
DrawLines
DrawDotLines
DeleteBox
MoveBox
Draw
IdentifyBox
RefreshBoxes
DrawRectangle
DrawRectangle2

Figure 3.c

E. HOW TO PRESENT THE MENU

The library function *PresentMenu(menu,answers)* presents all the menus in the program. *PresentMenu* displays a Pop-up menu described by the structure pointed to by menu, interacting with the user to obtain a result. The initial state of the menu is defined by the array of pointers *answers*. "Answer", "Attr_answer", "Top_Answer" are the examples in the program.

A menu is defined as a sequence of questions, each question consisting of various choices. The data structure representing a menu is a tree of structures. At the root of the tree there is a single *struct menu* structure that consist of three components: *label, size, question. label* is a pointer to a string that is used as the title of the menu. *Welcome to DB world. Help. DB Name. OBJ ENTRY, GENOBJ ENTRY, Attributes* are the examples used in the program. The title of the menu is centered in the title bar of the menu window.

Size indicates the number of questions in the menu. In the program there are different types of menus asking different number of questions. The top menu has only one question consisting of four choices. The *Help* menu also asks one question consisting of three choices. *db_name_enter_menu* which asks the name of the database to be dealt with, has only one question. The menu *dbname_save_menu* asks two questions: One is the name of the DB to be dealt with and the other one is the question that prompts whether the user wants to continue to perform. The menu *object_entry* asks six question. The first one is the name of the object and the rest is the names of the object that this particular

object has a relation. The menu *genobj_menu* asks 7 questions: the first one is the name of the database. the next three question are the names of the relations and the rest is the names of the subobjects. The menu *attribute_menu* asks 6 questions that are the names of the attributes of an object.

Questions appear in the menu in the order in which they appear in the array. A question structure has four components: *type*, *label*, *size*, *choices*. *type* is the type of the question being used in the program. They are SELECT, STRING and TOGGLE. SELECT question is one in which the choices are mutually exclusive. exactly one of the possible choices may be selected by the user at any time. A STRING question is one in which the result is a string of text entered by the user. Type TOGGLE has not been used in the program.

label is a pointer to a string which will appear beside the question. Size indicates the number of choices in the question: (For example. the question *top_entry* has 4 choices.) Or in the case of string questions. the maximum length of the string result.

Choice structure has three components: *type*, *label*, and *shade*. *type* is the type of the choice which is only label. *label* is a pointer to a string which will appear in the choice. This is the string at which users point to make a selection. *shade* indicates the background color of the box representing the choice.

For each question the corresponding pointer on arrays points to the default value for the question and, on return, to the result given by the user. For a select question the pointer is to an integer whose value is the position of the selected

choice in the choice array for the question. For a string question, the pointer is to an array of characters representing the string.

The first menu is the top level menu. The name of the menu is *top_menu*. The title of the menu is "Welcome to DB world. There is only one question to be asked, which is one element array with name *top_entry*. The type of the question is SELECT. "Please_enter" appears to the left of the question. There are 4 choices to be selected and the array *top_choices* includes those four choices. The type of each choice is LABEL, the titles of the choices are the ones mentioned in the top level menu, and the color of the background is white, which is indicated with VT_White.

Each choice in the menu corresponds to an integer from 0 to 3. The answers taken from the menu is held in a variable. The main program continues working until this variable becomes 3 which corresponds to function *Quit*. In case the variable is zero *Defwindow* is operated. The window opened first is used as a *Define DB* window by changing its title.

Library function *SetWindowTitle(fd,title)* handles the problem of changing the title of the window. The way this function is used as follows: *SetWindowTitle* changes the title of the window associated with a specific file descriptor to the null terminated string pointed to by title . Title may have at most 31 non_null characters which are always painted in a standard title font and are automatically centered in the title bar of the window. The next usages of this window follows the same concept.

There is another window opened with title "DATA" which is used to enter the data for the database. This window is opened at the same time that the *Define DB* window is opened. The way the rest of the windows are opened in the program is as follows.

Open window(x,y,w,h,title) opens a window on the desktop. x indicates the x coordinate . y indicates the y coordinate of the window's upper left corner. h indicates the height and w indicates the width of the window, *title* "DATA" is the title of the window.

If the window already exists, by changing the window depth, existing window is reused. The library function *ChangeWindowDepth(fd,window,depth)* is used for this purpose.

ChangeWindowDepth changes the depth of the window according to its file descriptor to the depth given by the user. The number of the windows opened is always counted. Variable *dcount* counts the number of the windows that will be used recalling the windows. Zero depth of the window indicates that the window is the one currently being interacted. When the interaction is comlete the window can be put to the lowest levelof the windows, next window after it comes to the view and interaction is done with this window. By changing the window depths, every window that has been opened before can be reused. This method is followed in the entire program.

Function *Defwindow* also utilizes library function "PresentMenu" to ask the name of the database. The answer is saved in an array. Defwindow calls three

functions: *Initialize*, *TouchBoxes*, *Finish* which perform the actions in definition window. Each function will be detailed later in this chapter. When exiting the window, another menu prompting the name of the database and with the question asking whether to continue, is presented. The answer concerning to continue is kept in a variable *con*. *Define DB* window stays active until *no* answer has been received from the menu.

F. MANIPULATION WINDOW

Function *Manipwindow* does not call any function. Since it is only an initiative to DB manipulation, by using the library functions, it creates a blank window that will be used by the manipulation part. This window also uses menus concerning the name of the database to be dealt with and to save the diagram created for the database schema and whether to continue the actions in the window. Window stays active until *Exit* command has been received from the pop_up menu.

G. HELP WINDOWS

Function *Help* calls two functions: *Help1* and *Help2*. *Help1* is used for opening the window containing the information about how to define the database and *Help2* is used for opening the window about the manipulation of the database. These two windows are also blank windows to be filled out later with the information. Since the goal of this study is to use a window for the creation

of the database schema there has not been paid to much attention to these *Help* windows.

H. DESIGN OF THE DEFINITION WINDOW

Initialization of the window in done by function *Initialize*. *Refresh*, *RefreshBoxes* and *Background* are called by this function. *Refresh* is called as a parameter to a library function *SetRefresh*. Function *Refresh* also calls *BackGround* and *RefreshBoxes*. What *Background* and *RefreshBoxes* do will be explained later.

There are two types of line disciplines: NTTYDISC and TWSDISC. First one is a default line discipline. Second one is is the one for window graphics discipline. Function *Initialize* sets the line discipline to TWSDICS.

Another library function *SetMouseMode* sets the mouse mode for mouse input to VT_MOUSE_DOWN, meaning input is taken when mouse is pressed.

Thickness of all lines are initialized to BOX_LINE. That means, all lines drawn on the window will have two pixel long width.

I. BACKGROUND OF THE DEFINITION WINDOW

Background is the function that puts every detail on the window that enables user to use the window. The *schema design area*, *mode* boxes that represents the actions and *type* boxes representing objects(rectangles) and lines(associations) are put by this function.

There are some main tools that *Background* use. The first important one is current position pointer. Current position pointer points to any point on the window. To draw a rectangle border, to paint a rectangle, to draw a line, to paint a string is done first putting this pointer to the upper left corner of the candidate object and then doing the action. The library function to put the current position pointer to a certain place is *SetPosition*. Second important tool is the *CurrentColor*. The current color can be set to VT_WHITE or VT_BLACK or between white and black specifying the numbers corresponding to different tones of color gray.

By setting the color to a specific color and by putting the current position pointer to a certain point, a rectangle border can be drawn by library function *PaintRectangleBorder*, a rectangle interior can be painted by *PaintRectangleInterior*, a line can be drawn by *PaintLine*, a string can be painted by *PaintString*. Tool *SetJustification* centers the string relative to current position pointer.

Another library function *InvertRegion* is used for highlighting a specific region. To indicate that current *mode* and *type*, the white areas are inverted to black and black areas are inverted to white in the box. Every time one of the boxes is touched the box is inverted to indicate the current action being performed.

Refresh Routine, called by library function *SetRefresh*, gets the current permanent clipping bounds of the window. Clipping bounds means that any

object or part of an object drawn outside of this area is not displayed. First time window is opened clipping bounds covers the entire window. This clipping bounds can be set to a specific area of the window.

Library function *GetPermanentClipping* gets x and y coordinates and the width and height of the window so when function *Background* is called, the current position pointer is set to the upper left corner of the window, meaning that x and y coordinates are set to zero. Upper left corner of the window is used as a reference point when using the rest of the window, instead the upper left corner of the screen.

Refresh Routine calls function *Background* and *RefreshBoxes*. What *Background* does was explained before. What *RefreshBoxes* does as follows.

J. UPDATE THE RECTANGLES

RefreshBoxes gets the current permanent clipping bounds and restricts this area to the *schema design area*. This way the only area which can be drawn is the *schema design are*. First, the area is painted to white and then, if there is, the rectangles are drawn.

The features of Rectangles drawn in this area is kept in an array of records called as "rectangle". x,y coordinates and width and height of the rectangle, the name of the rectangle, the relation names of this rectangle, if it is a genobj, the subobject names of the rectangle are kept in this structure. Also whether it has

got a circle, representing disjunctive relation, in it and whether it is a rectangle representing a generalized object, is kept in this structure.

Every time *RefreshBoxes* is called the rectangles featuring the information on objects are drawn on the *schema design area*, one by one, until the rectangles are finished.

K. USE OF THE MOUSE TO CREATE AN OBJECT

What *TouchBoxes* does is to get the input from the mouse and do the appropriate action. In case the right most button is pressed and released it redoes the last action performed.

If left most button is pushed, the actions are different depending on the place of the mouse. The information about where mouse is put is received by function *UseableCoordinates*. If the mouse is in the *schema design area* it does the action depending on which mode is being used. If *Draw* mode is being used, a rectangle can be drawn: if mode is being used, a rectangle can be moved to another place in the schema design area: if *Delete* mode is being used, it deletes the existing rectangle pointed by the mouse.

In case the *Draw* mode is selected, 4 types of action can be done depending on current type. If current type is OBJ_TYPE, meaning that OBJECT box is selected, then function *DrawBox* is called which draws a box in the schema design area. In case GEN_OBJ type, the same action is done. But function *DrawBox* separates objects and generalized object depending on the boolean variable

genbool. In *Draw* mode, the lines and dashed lines can also be drawn. If the circlebox is touched, boolean *circbool*, in the structure rectangle, becomes TRUE and the rectangle is drawn with a circle in it.

In case the middle mouse is pushed, a pop_up menu appears by library function *DisplayPopup*, it gives the capability to the user to exit the window or to clear the *schema design area*. If the *clear* command is chosen the action is to paint the drawing area to white, if *exit* command is chosen, then action is to exit the window and save the diagram.

L. DRAW A REGULAR OR NESTED RECTANGLE

Functions *DrawRectangle* and *DrawRectangle2* are called by functions *RefreshBoxes*, *DrawBox*, and *MoveBox*. As I explained above, *RefreshBoxes* draws all the rectangles in the array of records "rectangle" by calling these two functions. If the rectangle represents an aggregate object, then *DrawRectangle* is called, or if it is a generalized object, then *DrawRectangle2* is called. Both functions take one rectangle at a time and draw the rectangle depending on the information it contains. For example, if the *circbool* is TRUE than rectangle is drawn with a circle in it.

At the time when *DrawBox* is called by *TouchBoxes*, to draw the boxes, function *DrawBox* calls these functions, above, by sending the a record with every information in it. If *MoveBox* calls these two functions, the structure element

which is being moved, is sent to these functions for the next place of the rectangle.

M. DETERMINE WHERE CURRENTLY THE MOUSE IS

Function *UseableCoordinates* determines where currently the mouse is. If it is the *schema design area*, then drawing, moving, or deleting can be done. If it is in the *type boxes* area, then it sets the types to OBJECT, GENOBJ, LINE or DASHEDLINE by inverting their boxes. If the mouse is inside the circle box then it sets the boolean variable of the record to TRUE so when drawing the rectangle it draws with a circle in it. If it is put inside the mode boxes, it sets the mode to one of three modes, and inverts its box.

N. IDENTIFY A RECTANGLE(OBJECT)

Function *IdentifyBox* is called by functions *MoveBox* and *DeleteBox*. When moving or deleting the box, the mouse is put into the rectangle which is to be moved or deleted. What *IdentifyBox* does is that it identifies which record of the structure "rectangle" that the rectangle is, and returns the ID # of the record.

O. MOVE A RECTANGLE(OBJECT)

Function *MoveBox* gets the ID # of the rectangle to be moved and tracks the action of the mouse. When the mouse is released the data about the new place of the rectangle is put as a last record of the array and the previous information is

deleted. By calling RefreshBoxes. the array of records "rectangle" is redrawn with the new changes in it.

P. DELETE A RECTANGLE(OBJECT)

Function *DeleteBox* also calls *IdentifyBox* to get the ID # of the rectangle to be deleted. And deletes the rectangle. The number of rectangles is decreased by one. By calling *RefreshBoxes* the place of the rectangle that is already deleted is painted to white and the entire array is redrawn.

Q. CREATE A RECTANGLE(OBJECT)

When function *DrawBox* is called. first. the number of the rectangles is increased then the x and y coordinates of the mouse is put to record's x and y coordinates. The data about the name of the rectangle(object). relations of the object and. if there is. subobjects of the object taken from the menu are put into the last element added to the array. If the right most button is pushed. then last element of the array will be redrawn. The width and the height of the last element is put to the added elements width and height part. If the left most button is pushed and the mouse is moved while it is being pushed. library function *TrackRubberBox* tracks the lower right corner of the rectangle. returning the width and the height of the new rectangle.

The movement of the mouse is considered positive from left to right and from up to down. If the rectangle is moved opposite to those above. then some arrangement is done concerning to change the negative values.

R. LINK ASSOCIATED OBJECTS WITH SOLID LINES

Function *Drawlines* takes the first record of the array and compares the relation names of this record with the rest of the object names. If there is a match it calls the function *Draw* to draw a line between them by sending the ID #'s of the matched elements and then takes the next element and continues until the elements of the array finishes.

S. LINK ASSOCIATED OBJECTS WITH DASHED LINES

Since the dashed lines in the schema represent a relation between an object and a subobject of an general object it compares the names of the relations and the names of the subobjects and calls the same function *Draw* to draw a solid line. Because there is no library function to draw a dashed line.

Function *Draw* takes the ID #'s of two object, which there is a relation between them. and draws a line between them depending on the position of two rectangles. If a rectangle is below the other rectangle, it draws a line from the middle of the bottom line to the middle of the upper line of the other rectangle. If they are almost the same level, it draws a line from right side of one rectangle to the left side of the other rectangle.

VI. CONCLUDING REMARKS

The entire thesis has been designated to the implementation of GLAD's data definition language. The program that is capable of having the user define a database schema, has been coded in programming language C. The system that the implementation was achieved on has a C language graphics library. This library enabled us to write the program and utilize the graphics capability of the system.

Implementation of the program has been explained in Chapter 5. The usage of the library functions and some of the important points of the library functions have been explained, as well. The entire library can be found in the manual of "isiv". This manual provides information on how to use the system, how to write graphics programs by using the library functions. This manual also provides some example programs that show how the library functions can be used to write graphics programs.

Chapter 1 explains the major components of a database management system which they are data definition language and data manipulation language. Some query languages have been used for this purpose. These languages brought some problems with them. The difficulties using these languages forced researchers to take advantage of the new graphics capabilities of the computers. Chapter 2

explains the difficulties in using the query languages and prior researches to overcome those difficulties.

GLAD's approach to the issue is to use a diagram representing the database schema as an interface to the database. Over the past years, database schema has been used to explain the database concepts. But, it has not been used in accessing the databases. By visualizing the database concepts, user can understand the underlying concepts of the database.

Implementation of the GLAD's data definition language can be considered as a prototype. It is not complete, in terms of representing the entire data definition language. But, it gives an idea, how computer graphics can be utilized to access the database, specifically to define a database. When it is complete, I believe, it will be rather useful to the database users.

APPENDIX - PROGRAM LISTING

```
/* this program is written for the data definition language
  of GLAD(Graphical Language in Accessing Data bases).*/
#include <vt.h>
#include <tools.h>
#include <bitmap.h>


#define MAX_RECTANGLE      100 /* max rect.(objects) in the schema area */
#define SPACE               10
#define BOX_LINE             2   /* max width of all the lines */
#define BOX_WIDTH           50
#define BOX_HEIGHT          50
#define NTYPES               4

#define TYPE_BOX_X        SPACE  /* x coord. of type boxes */
#define TYPE_BOX_Y        SPACE  /* y coord. of type boxes */
/* width and height of the type boxes */
#define TYPE_BOX_W        (NTYPES*2*BOX_WIDTH)
#define TYPE_BOX_H        BOX_HEIGHT

/* x and y coordinates of the circle box */
#define CIRC_BOX_X        (TYPE_BOX_X+TYPE_BOX_W + SPACE)
#define CIRC_BOX_Y        SPACE
#define CIRC_BOX_W        BOX_WIDTH   /* width of the circle box */
#define CIRC_BOX_H        BOX_HEIGHT  /* height of the circle box */

#define OBJECT_TYPE       4
#define GENOBJ_TYPE       5
#define LINE_TYPE         6
#define DOTLINE_TYPE      7

#define NMODE             3

/* x and y coordinates of the mode boxes */
#define MODE_BOX_X        (TYPE_BOX_X+TYPE_BOX_W+100
                              +BOX_WIDTH)
#define MODE_BOX_Y        SPACE
/* width and height of the mode boxes */
#define MODE_BOX_W        (NMODE*2*BOX_WIDTH)
#define MODE_BOX_H        BOX_HEIGHT
```

```
=define DRAW_MODE        0
=define MOVE_MODE        1
=define DELETE_MODE      2

/* x and y coordinates of the schema design area */
#define DRAW_BOX_X       (TYPE_BOX_X)
#define DRAW_BOX_Y       (TYPE_BOX_Y+TYPE_BOX_H+SPACE)


struct wstate   istate;  /* for the current state of the window */
struct vtseq    input;   /* for the input being received from the mouse */
struct rectangle {       /* for the data on the rectangles(objects) */
        short x;       /* x coord..of the rectangle(object) */
        short y;       /* y coord. of the rectangle(object) */
        short w;       /* width of the rectangle(object) */
        short h;       /* height of the rectangle(object) */
        char  name[20]; /* name of the object written in the rectangle */
        /* relation names of the object */
        char  rel1[20];
        char  rel2[20];
        char  rel3[20];
        char  subobj1[20]; /* subobject names of the object,if the object*/
        char  subobj2[20]; /* is an generalized object.              */
        char  subobj3[20];
        bool  genbool;     /* TRUE for a generalized object          */
        bool  circbool;    /* TRUE for a rectangle with circle in it */
            } dlist[MAX_RECTANGLE]; /* array of rectangles(objects)  */

/* for the width and height of the drawing area depending on
   the size of the window */
short   DRAW_BOX_W,DRAW_BOX_H;
short   npoints      = -1;    /* keeps track of the number of rectangles */
short   current_mode = DRAW_MODE;
short   current_type = OBJECT_TYPE;
short   repeat       = 0;     /* for a rectangle to be redrawn */

/* top level menu that appears when first DB window created and after
   finishing to use one of the top level windows. */
/* this is an array choices, each element of the array having three components:
   type of the choice which they are LABEL:label of the choice which appears
   in the box and background color of the choice which they are all VT_White,
   meaning that the background color of the labels are white.          */
struct  choice top_choices[] = {
```

```
        {LABEL.     " Define DB ".        VT_White}.
        {LABEL.     " Manipulate DB ".    VT_White}.
        {LABEL.     "   HELP   ".         VT_White}.
        {LABEL.     "   QUIT   ".         VT_White} };
/* this is an array of questions.each element having four  components :
   type of the question is SELECT.meaning that exactly one of the possible
   choices may be selected by the user at any time.second component is the
   label of the question. there are 4 choices to present and the name of
   the array of the choices is the last one                              */
struct  question top_entry[] = {
        {SELECT,    "Please Enter", 4,  top_choices}
};

/* this structure consists of three components : title of the menu window,
   1 questions to be asked and the name of the array of the questions.     */
struct  menu top_menu = {
        "Welcome to DB World",  1,  top_entry
};


/* this is an array of choices for the question help_entry.each element of the
   array having three components: type of the choice which they are LABEL,the
   title of the choice.and the background color of the choice.            */
struct  choice help_choices[] = {
        {LABEL,     " DB definition ",      VT_White},
        {LABEL,     " DB manipulation ",    VT_White},
        {LABEL,     "   QUIT   ",          VT_White}
};

/* this is an array of questions for help_menu.this array has only one element
   in it and this element has three components: type of the question(SELECT),
   label of the question.number of the choices(3).and the name of the array
   of the choices.                                                        */
struct  question help_entry[] = {
        {SELECT,   "Description ",   3, help_choices}
};

/* this structure has three components : title of the menu window,
   1 questions to be asked and the name of the array of the questions.     */
struct  menu help_menu = {
        "HELP",             1,   help_entry
};
```

```
/* this is an array of questions for the menu "dbname_enter_menu".it has only
   one element in it and this element consists of four components : the first
   component STRING is the type of the choice which is a question in which
   the result is a string of text entered by the user.second one is the label
   for the string.25 indicates the max. number of letters alloved for the
   string also 0 indicates that there is not an array of choices for string. */
struct question name_enter[] = {
        {STRING,  "Please Enter Your DB Name", 25,0},
};

/* this menu asks the name of the data base.it has three components:the title
   of the window is "DB Name".there is only one question to be asked.the name
   of the array of the questions is name_enter.
struct menu dbname_enter_menu = {
        "DB Name",          1,    name_enter
};


/* this is an array of choices for the second question of base_name        */
struct choice cont_choices[] = {
        {LABEL,   " yes ",              VT_White},
        {LABEL,   " no ",               VT_White}
:

/* this is an array of the questions for menu "dbname_save_menu".first element
   is a string holding the name of the data base entered by the user when menu
   "dbname_enter_menu" was presented.it has four components:STRING,type of
   the question:label of the question:25,max number of letters in the string:
   0,indicating that there is not an array questions for the string.
   second element of the array has four components : SELECT,type of the
   question:label of the question:2,number of the choices and the name
   of the array of the choices.                                          */
struct question base_name[] = {
        {STRING,  "Your DB is saved with name",25,0},
        {SELECT,  "Continue ?",2,     cont_choices}
};

/* this structure has three components: the title of the menu window is
   "DB name";there are 2 questions to be asked and the array of questions  */
struct menu dbname_save_menu = {
        "DB name",          2,    base_name
};
```

```
/*  this is an array of questions for the menu "object_entry".first element of
    the array asks the name of the object.the rest of the array ask the names
    of the.other object that this object has a relation.the first components
    are the type of question tobe asked.second ones are the labels for questions
    20 is the max number of letters in the string.                    */
struct  question obj_entry[] = {
        {STRING,  "Enter Your Object Name",  20, 0},
        {STRING,  "Enter The Relations   ",  20, 0},
        {STRING,  "                      ",  20, 0},
        {STRING,  "                      ",  20, 0},
        {STRING,  "                      ",  20, 0},
        {STRING,  "                      ",  20, 0}
};

/*  this menu has three components : "OBJECT ENTRY" is the title of the
    menu window.there are 6 questions to be asked.the array which is
    holding the questions is obj_entry. */                            */
struct  menu object_entry = {
        "OBJECT ENTRY",     6,     obj_entry
};


/*  this is an array of questions for the menu "genobj_menu".first question asks
    the name of the general object.from second to fourth ask the name of the
    other object that this object has relation.the rest of the questions are
    the subobject names of the general object. */
struct  question genobj_entry[] = {
        {STRING, "Enter Gen. Object name",    20, 0},
        {STRING,  "Enter The Relations ",     20, 0},
        {STRING,  "                    ",     20, 0},
        {STRING,  "                    ",     20, 0},
        {STRING,  "Enter Sub_Objects   ",     20, 0},
        {STRING,  "                    ",     20, 0},
        {STRING,  "                    ",     20, 0}
};

/*  this structure presents a menu for a general object. first component of the
    menu is the title of the menu window.  second one is the number questions
    appearing in the menu.  third one is the array holding the questions */
struct  menu genobj_menu = {
        "GENOBJ ENTRY",   7,     genobj_entry
};
```

```
/* this is an array of questions for the menu "attribute_memu". questions
   ask the attributes of an aggregate object. */
struct question attrib_entry[] = {
      {STRING, "Enter The Attributes ", 20, 0},
      {STRING, "                     ", 20, 0},
      {STRING, "                     ", 20, 0},
      {STRING, "                     ", 20, 0},
      {STRING, "                     ", 20, 0},
      {STRING, "                     ", 20, 0}
};

/* this structure presents a menu for the attributes of an aggregate
   object. the name of the menu window is "ATTRIBUTES" there are 6 attribute
   names to be asked and the questions is being held in the array
   attrib_entry */
struct menu attribute_menu = {
      "ATTRIBUTES",     6,     attrib_entry
};


int   top      = 0; /* variable holding top level answer:initialized to
                       zero indicating that the first one of the
                       choices(DB Definition) is the default choice */
int   help     = 0; /* variable holding the answer from help menu;
                       initialized to zero indicating that the first one of
                       the choices(Describe DB def.) is the default choice*/
int   con      = 0; /* variable holding the answer from dbname_save_menu:
                       initialized to zero indicating that the first
                       one of the choices(yes) is the default choice */

char  name[25]       = " "; /* variable holding the name of the data base */
char  dummy[25]      = " "; /* variable to clear the name in the
                                                  array "name" */
int   *top_answer[]  = {&top}; /* answer from the top level menu */
int   *help_answer[] = {&help}: /* answer from the help menu */
int   *dbname_answer[] = {(int *)name,&con}; /* take the name of the data */
                  /* base and answer as to continue to deal with it */

char  objname[20]= " ";
char  ans1[20]   = " ";
char  ans2[20]   = " ";
char  ans3[20]   = " ";
char  ans4[20]   = " ";
```

```
char    ans5[20]  = " ":
char    ans6[20]  = " ":
char    ans7[20]  = " ":
char    ans8[20]  = " ":
char    ans9[20]  = " ":
char    ans10[20] = " ":

char    att1[20] = " ";
char    att2[20] = " ";
char    att3[20] = " ";
char    att4[20] = " ";
char    att5[20] = " ";
char    att6[20] = " ";

/* holds the object name. the relation names and the subobject names of
   a general object. */
int     *answer[] = {
          (int*)objname,(int*)ans1,(int*)ans2,
          (int*)ans3,  (int*)ans4,(int*)ans5,
          (int*)ans6,  (int*)ans7,(int*)ans8,
          (int*)ans9,  (int*)ans10
};

/* holds the attirbute names of an aggregate object */
int     *attr_answer[] = {
          (int*)att1,(int*)att2,(int*)att3,
          (int*)att4,(int*)att5,(int*)att6
};


int     fd;
int     f = 1;
int     count;
int     dcount;      /* window number of the definition window */
int     mcount;      /* window number of the manipulation window */
int     h1count;     /* window number of the describe definition window */
int     h2count;     /* window number of the describe manipulation window */
bool    boolcirc  = FALSE; /* whether the circle box has been touched */
bool    touchbool = FALSE;
short   option;      /* for the pop-up menu */
short   defwinum  = 0; /* to find out if definition window already exists */
short   manwinum  = 0; /* to find out if manip. window already exists */
short   helpdefnum = 0; /* to find out if describe defin. window exists */
```

```
short   helpmannum = 0; /* to find out if describe manip. window exists */
short   blackobj   = 0; /* to invert one of the type boxes */
main()
{
   count = 2;

   /* continue until variable top becomes 3.representing "Quit" */
   while (top != 3) {

      /* present the top level menu and get the answer */
      PresentMenu(&top_menu.top_answer);

      /* if the answer.from the menu,is "DB Definition" */
      if (top == 0) {

         /* change the window title to "DB Definition" */
         SetWindowTitle(f."DB Definition");

         if (defwinum == 0) {

            /* open a window for the data entry */
            OpenWindow(550.600.678.332."DATA ");

            count = count + 1;
            dcount = count;
            Defwindow();
         } /* end if */

         else {
            /*get the "DB Definition" window by changing the window depth */
            ChangeWindowDepth(f,dcount.0);
            Defwindow();
         } /* end else */

      } /* end if */

      /* if the answer.from the menu,is "DB Manipulation" */
      if (top == 1) {

         /* if "DB Manipulation" window does not exist yet then open the
            window and call the manipwindow */
         if (manwinum == 0) {
            OpenWindow(118,117,878.432,"DB Manipulation");
```

```
            count  = count + 1;
            mcount = count;
            Manipwindow();
         } /* end if */

         /* if the "DB Manipulation" window already exists then get the
            window by changing the window depth.                    */
         else {
            ChangeWindowDepth(f,mcount,0);
            Manipwindow();
         } /* end else */

      }

      /* if the "Help" option is selected */
      if (top == 2)

         Helpwindow();
   } /*  end while  */

   Quit();

} /* end main */


Quit()
{
   GetWindowState(f,&istate);
   fd = -1;
   SetWindowState(f,&istate);
} /* end function */


Defwindow()
/* this function is about "DB Defintion" window */
{
   /* get the current state of the window */
   GetWindowState(f,&istate);

   defwinum = defwinum + 1;

   /* ask the name of the data base and get the answer */
```

```
    PresentMenu(&dbname_enter_menu.dbname_answer);

    /* continue until "no" answer has been received */
    while (con != 1 ) {

        /* initializa the window */
        Initialize();

        /* do the actual creation of the data base schema */
        TouchBoxes();

        /* close the window */
        Finish();

        /* present the name of the data base and ask whether to continue */
        PresentMenu(&dbname_save_menu.dbname_answer);

        SetWindowTitle(f,"DATABASE");
    } /* end while */

    /* if "no" answer is received then change the window depth */
    if (con == 1) {
        ChangeWindowDepth(f,2,0);
        con = 0;
    } /* end if */

    /* clear the data base name in the array "name" for another entry */
    copy(dummy.name);
} /* end function */



Manipwindow()
/* this function handles the "DB Manipulation" window.only a blank window is
   created for the completeness of the GLAD */

{
    manwinum = manwinum + 1;

    /* get the current state of the window */
    GetWindowState(f,&istate);

    /* ask the name of the data base and get the answer */
```

```
   PresentMenu(&dbname_enter_menu.dbname_answer):
   option = -1:

   /* continue until "exit" command is received or "no" option is selected */
   do
   {
      option = DisplayPopUp(1,"menu  EXIT  SAVE  ")-1;

      /* if "exit"command is received from the pop-up menu */
      if (option == 0) {
         ChangeWindowDepth(f,2,0);
      }

      /* if "save"command" is received from the pop-up menu,then present
        the name of the data base currently being dealt with and get
        the answer as to continue to work */
      else if (option == 1) {
         PresentMenu(&dbname_save_menu.dbname_answer):
      }

   }
   while (con != 1 && option != 0);

   /* of "no" answer is received then change the window depth */
   if (con == 1) {
      ChangeWindowDepth(f,2,0):
      con = 0;
   } /* end if */

   /* clear the data base name in the array "name" for another entry */
   copy(dummy,name);
} /* end function */



Helpwindow()
{
   /* ask what kind of information the user wants and get the answer */
   PresentMenu(&help_menu.help_answer):

   /* continue until "Quit" option is received. */
   while (help != 2 ) {
```

```
    /* if "Describe DB Defintion" is selected */
    if (help == 0 ) {

       /* if the window does not exist yet.then open the window */
       if (helpdefnum == 0) {
          OpenWindow(118,117.878,432,"DESCRIBE DB DEFINITION");
          count   = count + 1;
          h1count = count;
          Help1();
       } /* end if */

       /* if the window already exists.the change the window depth and
          get the window */
       else {
          ChangeWindowDepth(f.h1count,0);
          Help1();
       } /* end else */
    } /* end if */

    /* if "Describe DB manipulation " is selected */
    else if (help == 1) {

       /* if the window does not exist yet,then open the window */
       if (helpmannum == 0) {
          OpenWindow(118,117.878,432,"DESCRIBE DB MANIPULATION");
          count   = count + 1;
          h2count = count;
          Help2();
       } /* end if */

       /* if the window already exists,the change the window depth and
          get the window */
       else {
          ChangeWindowDepth(f,h2count,0);
          Help2();
       } /* end else */

    } /* end else */

  } /* end while */

  help = 0;
} /* end function */
```

```
Help1()
{
   helpdefnum = helpdefnum + 1:
   option = -1;

   /* display the window until "exit"command is
     received from the pop-up menu*/
   while (option != 0) {
      option =  DisplayPopUp(1,"menu  EXIT  ")-1;
   } /* end while */

   /* if the "exit" command is received, then change
     the window depth and present the "Help" menu  */
   ChangeWindowDepth(f,2,0);
   PresentMenu(&help_menu,help_answer);

} /* end function */



Help2()
{
   helpmannum = helpmannum + 1;
   option = -1;

   /* display the window until "exit"command is
     received from the pop-up menu */
   while (option != 0)  {
      option = DisplayPopUp(1,"menu  EXIT  ")-1;
   }

   /* if the "exit" command is received, then change the window depth and
     present the "Help" menu                                          */
   ChangeWindowDepth(f,2,0);
   PresentMenu(&help_menu,help_answer);

} /* end function */



copy(s1,s2)
char s1[],s2[];
{
```

```
    int i;

    i = 0;
    while ((s2[i] = s1[i]) != ' ')
        ++i;

} /* end function */


Initialize()
{
    void Refresh(),RefreshBoxes();

    /* get the current state of the window */
    GetWindowState(f,&istate);

    /* block asynchronous refresh and adjust for the window */
    BlockRefreshAdjust(1);

    /* specify and identifier and a refresh routine for the window */
    SetRefresh(f,0,Refresh);

    /* set the line discipline of the window to graphics line discipline */
    SetLineDisc(f,TWSDISC);

    /* allocate a buffer size of 1024 for the window */
    SetBuf(f,1024);

    /* inform the application when the mouse button is released */
    SetMouseMode(f,VT_MOUSE_DOWN);

    /* set the thickness of the lines and objects borders
       to BOX_LINE( =2 )
    SetThickness(f,BOX_LINE);

    /* adjust the width and height of the schema design area
       according to the with and height of the window          */
    DRAW_BOX_W = istate.width - DRAW_BOX_X - (SPACE*2);
    DRAW_BOX_H = istate.height- DRAW_BOX_Y - (SPACE*2);
    Background();
    RefreshBoxes(DRAW_BOX_X-BOX_LINE,DRAW_BOX_Y-BOX_LINE,
         RAW_BOX_W+(2*BOX_LINE),DRAW_BOX_H+(2*BOX_LINE));
```

```
} /* end function */


Background()
{
   short  i;

   /* set the current position pointer to the upperleft corner of the window */
   SetPosition(f,0,0);

   /*set the current foreground color to white */
   SetColor(f,VT_White);

   /* paint the entire window to white */
   PaintRectangleInterior(f,istate.width,istate.height);
   SetPosition(f,TYPE_BOX_X,TYPE_BOX_Y);
   PaintRectangleInterior(f,TYPE_BOX_W,TYPE_BOX_H);

   SetPosition(f,CIRC_BOX_X,CIRC_BOX_Y);
   SetColor(f,VT_White);
   PaintRectangleInterior(f,BOX_WIDTH,BOX_HEIGHT);

   /*set the current foreground color to black */
   SetColor(f,VT_Black);

   /* set current position pointer to the upperleft corner of the type boxes */
   SetPosition(f,TYPE_BOX_X,TYPE_BOX_Y);

   /* draw the border of type boxes */
   PaintRectangleBorder(f,TYPE_BOX_W,TYPE_BOX_H);

   /* draw the vertical lines to separate different types */
   for (i=(TYPE_BOX_X + 2*BOX_WIDTH);
       i<(TYPE_BOX_X+TYPE_BOX_W); i+=2*BOX_WIDTH) {
      SetPosition(f,i,TYPE_BOX_Y);
      PaintLine(f,0,BOX_HEIGHT);
   } /* end for */

   /* center the text relative to position pointer */
   SetJustification(f,VT_CENTER);

   /* set current position pointer in the middle of the object type box */
```

```
SetPosition(f,(TYPE_BOX_X+BOX_WIDTH),
        (TYPE_BOX_Y+(TYPE_BOX_H/2)));

/* draw the text "OBJECT" in the first box */
PaintString(f,VT_STREND,"OBJECT");


/* set current position pointer into the genobj type box */
SetPosition(f,(TYPE_BOX_X+(2*BOX_WIDTH+6)),(TYPE_BOX_Y+5));

/* draw the second rectangle in the genobj type box */
PaintRectangleBorder(f,(2*BOX_WIDTH-12),(BOX_HEIGHT-10));

/* set current position pointer into the center of the genobj type box */
SetPosition(f,(TYPE_BOX_X+(3*BOX_WIDTH)),
        (TYPE_BOX_Y+(TYPE_BOX_H/2)));

/* draw the text "GENOBJ" in the second box */
PaintString(f,VT_STREND,"GENOBJ");

/* set current position pointer to the upperleft corner of the circle box */
SetPosition(f,CIRC_BOX_X,CIRC_BOX_Y);

/* draw the rectangle of circle box */
PaintRectangleBorder(f,CIRC_BOX_W,CIRC_BOX_H);

/* set current position pointer in the third line type box */
SetPosition(f,(TYPE_BOX_X+210),(TYPE_BOX_Y+(TYPE_BOX_H/2)));

/* draw a straight line in this box */
PaintLine(f,80,0);

/* center the text relative to position pointer */
SetJustification(f,VT_CENTER);

/* set current position pointer in the center of the dashedline type box */
SetPosition(f,(TYPE_BOX_X+(7*BOX_WIDTH)),
        (TYPE_BOX_Y+(TYPE_BOX_H/2-4)));

/* draw the string in the box */
PaintString(f,VT_STREND,"_______");

/* set current position pointer in the center of the circle box */
```

```
SetPosition(f,(CIRC_BOX_X+BOX_WIDTH/2),
        (CIRC_BOX_Y+BOX_HEIGHT/2));
/* draw a circle in the circle box */
PaintCircleBorder(f,4);

/* if one of the type boxes is touched then invert the box */
if (touchbool == TRUE)
   InvertRegion(f,(TYPE_BOX_X+blackobj*(2*BOX_WIDTH)),
            TYPE_BOX_Y,(2*BOX_WIDTH),TYPE_BOX_H);

/* set current position pointer to the upperleft corner of mode boxes */
SetPosition(f,MODE_BOX_X,MODE_BOX_Y);

/* draw the border of the mode boxes */
PaintRectangleBorder(f,MODE_BOX_W,MODE_BOX_H);

/* draw the vertical lines to separate the modes */
for (i=(MODE_BOX_X+(2*BOX_WIDTH));
    i<(MODE_BOX_X+MODE_BOX_W);i+=(2*BOX_WIDTH)) {
   SetPosition(f,i,MODE_BOX_Y);
   PaintLine(f,0,BOX_HEIGHT);
} /* end for */
/* center the text relative to position pointer */
SetJustification(f,VT_CENTER);

/* set current position pointer in the center of the "Draw" mode box */
SetPosition(f,(MODE_BOX_X+BOX_WIDTH),
        (MODE_BOX_Y+(MODE_BOX_H/2)));

/* draw the the string "Move" inside this box */
PaintString(f,VT_STREND,"Draw");

/* set current position pointer in the middle of the "Move" mode box */
SetPosition(f,(MODE_BOX_X+(3*BOX_WIDTH)),
        (MODE_BOX_Y+(MODE_BOX_H/2)));

/* draw the the string "Move" inside this box */
PaintString(f,VT_STREND,"Move");

/* set current position pointer in the middle of the "Delete" mode box */
SetPosition(f,(MODE_BOX_X+(5*BOX_WIDTH)),
        (MODE_BOX_Y+(MODE_BOX_H/2)));
/* draw the the string "Delete" inside this box */
```

```
    PaintString(f,VT_STREND,"Delete");

    /* invert the first box "Draw" mode to indicate that it is default mode */
    InvertRegion(f,(MODE_BOX_X+current_mode*(2*BOX_WIDTH)),
                MODE_BOX_Y,(2*BOX_WIDTH),MODE_BOX_H);

} /* end function */


void RefreshBoxes(x,y,w,h)
short   x,y,w,h;
{
    /* Each window has an associated rectangular area known as its clipping
       bounds.Any object or part of an object drawn outside this area is not
       displayed.                                                    */

    short cx,cy,cw,ch,i,a;

    /* return the X and Y coordinates of the upper left corner,width and height
       of the window that are the current clipping bounds            */
    GetPermanentClipping(f,&cx,&cy,&cw,&ch);

    /* specify the new rectangular area known as schema design area  */
    SetPermanentClipping(f,DRAW_BOX_X-BOX_LINE,DRAW_BOX_Y-
            BOX_LINE,DRAW_BOX_W+(2*BOX_LINE),
            DRAW_BOX_H+(2*BOX_LINE));

    /* restrict the current clipping bounds to the schema design area */
    RestrictPermanentClipping(f,x,y,w,h);

    /* set the current color to white */
    SetColor(f,VT_White);

    /* set current position pointer to the upper left corner of schema drawing
       area                                                          */
    SetPosition(f,x,y);

    /* paint the schema design area to white */
    PaintRectangleInterior(f,w,h);

    /* draw all the rectangles in the array of records "dlist";if the rectangle
       represents a generalized object then draw double rectangle    */
    for (i=0;i<=npoints;i++) {
```

```
        if (dlist[i].genbool == FALSE)
            DrawRectangle(&dlist[i]);
        else if (dlist[i].genbool == TRUE)
            DrawRectangle2(&dlist[i]);
        boolcirc = FALSE;
    } /* end for */
    /* set the current color to black */
    SetColor(f.VT_Black);

    /* set current position pointer to the upper left corner of the schema
       design area */
    SetPosition(f.DRAW_BOX_X,DRAW_BOX_Y);

    /* draw the rectangle border of the schema design area */
    PaintRectangleBorder(f.DRAW_BOX_W,DRAW_BOX_H);

    /* set current clipping bounds back to the entire window */
    SetPermanentClipping(f.cx,cy,cw,ch);
} /* end function */



void Refresh(id,x,y,w,h)
int    id;
short  x,y,w,h;
{
    short cx,cy,cw,ch;

    /* get X,Y coords. and width and height of the window for clipp bounds */
    GetPermanentClipping(f.&cx,&cy,&cw,&ch);

    /* set current clipping bounds the schema design area */
    SetPermanentClipping(f.x,y,w,h);

    /* draw the background */
    Background();

    /* draw the schema design area and the rectangles in it */
    RefreshBoxes(x,y,w,h);

    /* set current clipping bounds back to the entire window */
    SetPermanentClipping(f.cx,cy,cw,ch);
} /* end function */
```

```
Finish()
{
   /* set current clipping bounds to entire window */
   SetPermanentClipping(f,0,0,10000,10000);

   /* set current position pointer to the upper left corner of the window */
   SetPosition(f,0,0);

   /* set current color to white */
   SetColor(f,VT_White);

   /* paint the entire window to white  */
   PaintRectangleInterior(f,10000,10000);

   /* set current state of the window */
   SetWindowState(f,&istate);

   /* write out any data buffered for the window */
   Flush(f);
} /* end function */


TouchBoxes()
{
   short    x,y,w,h;

   /* continue until "exit" command is received from pop-up menu */
   for (;;) {
      /* get a single input sequence */
      switch(getvtseq(f,&input)) {

      /* if input is received via mouse */
      case VT_MOUSE :
          repeat = 0;
          /* which mouse button is used? */
          switch(input.u.mouse.buttons &
            VT_MOUSE_LEFT|VT_MOUSE_MIDDLE|VT_MOUSE_RIGHT)) {

          /* if the right most button is used then repeat the last action */
          case VT_MOUSE_RIGHT:
              repeat = 1;
```

```
/* if the left most button is used */
case VT_MOUSE_LEFT:

    /* if the mouse is inside the schema design area */
    if (UseableCoordinates()) {

        /* set clipping bounds to schema design area */
        SetPermanentClipping(f.DRAW_BOX_X,DRAW_BOX_Y,
                             DRAW_BOX_W,DRAW_BOX_H);
        /* check current mode */
        switch (current_mode) {

        /* if the current mode is draw mode...*/
        case DRAW_MODE:

            /* check the current type */
            switch (current_type) {

            /* if object box is selected,then draw a rectangle */
            case OBJECT_TYPE:
                DrawBox();
                break;

            /*if "GEBOBJ" box is selected,then draw double rect.*/
            case GENOBJ_TYPE:
                DrawBox();
                break;

            /* if straight line box is selected,then draw straight
              lines between rectangles                          */
            case LINE_TYPE:
                DrawLines();
                break;

            /* if dashed line box is selected,then draw dashed
              lines between rectangles                          */
            case DOTLINE_TYPE:
                DrawDashedLines();
                break;
            }
            break;

        /* if the current mode is "Move" mode,then move a
```

```
            box pointed by the mouse                    */
          case MOVE_MODE:
              MoveBox();
              break;

          /* if the current mode is "Delete" mode,then delete a
             box pointed by the mouse.
          case DELETE_MODE:
              DeleteBox();
              break;
          default:
              break;
          } /* end switch */
          /* set clipping bounds to the entire window */
          SetPermanentClipping(f,0,0,10000,10000);
      }
      break;

  /* if the middle button is used... */
  case VT_MOUSE_MIDDLE:

      /* set the current position pointer to the place that
         mouse currently stays                          */
      SetPosition(f,input.u.mouse.x,input.u.mouse.y);

      /* check the answer from the pop-up menu */
      switch(DisplayPopUp(f,"do clear Exit ")) {

      /* if the answer is "clear"... */
      case 1:
          npoints = -1;  /* no more records of rectangles */

          /* set current position pointer to the upper left corner
             of the schema design area                    */
          SetPosition(f,DRAW_BOX_X,DRAW_BOX_Y);

          /* set current color to white */
          SetColor(f,VT_White);

          /* paint schema design area to white */
          PaintRectangleInterior(f,DRAW_BOX_W,DRAW_BOX_H);
          break;
```

```
                case 2:
                    return;
                    break;
                } /* end switch(DisplayPopUp) */
                break;
            }
            break;
            /* if keyboard is used... */
            default:
                /* draw a status bar near the bottom of the screen */
                DisplayStatus(f,"use the mouse");
                break;
        } /* end switch(getinp) */

    } /* end for */

} /* end function */



/* this routine takes care of drawing a rectangle.which represents an aggregate
   object.with the name of the object in it . if the object has a disjunctive
   relation it draws the rectangle with a circle.X.Y coordinates and width.
   height of the rectangle.the name of the object and information wheter it
   has a disjunctive relation is received with the element of the structure
   "rectangle". */
DrawRectangle(r)
struct rectangle *r;
{
    /* set the current position pointer to the upper left corner of rect. */
    SetPosition(f,r->x,r->y);

    /* set the current color to white */
    SetColor(f,VT_White);

    /* paint inside the rectangle to white */
    PaintRectangleInterior(f,r->w,r->h);

    /* set the current color to black */
    SetColor(f,VT_Black);

    /* set the current position pointer to the upper left corner of rect. */
    SetPosition(f,r->x,r->y);
```

```
    /* draw the border of the rectangle */
    PaintRectangleBorder(f,r->w,r->h);
    r->genbool = FALSE;

    /* set current position pointer on the middle of the rectangle */
    SetPosition(f,(r->x+r->w/2),(r->y+r->h/2));

    /* center the string relative to the current position pointer */
    SetJustification(f,VT_CENTER);

    /* draw the name of the object in the rectangle */
    PaintString(f,VT_STREND,r->name);

    /* if object has a disjunctive relation ,then draw a small circle
       under the upper border of the rectangle                    */
    if (r->circbool == TRUE) {
        SetPosition(f,(r->x+r->w/2),r->y+3);
        PaintCircleBorder(f,4);
    } /* end if */
    boolcirc = FALSE;
} /* end fuction */


/* this routine takes care of drawing a rectangle,which represents a generalized
   object,with the name of the object in it .  if the object has a disjunctive
   relation it draws the rectangle with a circle.X,Y coordinates and width,
   height of the rectangle,the name of the object and information wheter it
   has a disjunctive relation is received with the element of the structure
   "rectangle". */
DrawRectangle2(r)
struct rectangle *r;
{

    /* set the current position pointer to the upper left corner of rect. */
    SetPosition(f,r->x,r->y);

    /* set the current color to white */
    SetColor(f,VT_White);

    /* paint inside of the rectangle to white */
    PaintRectangleInterior(f,r->w,r->h);
```

```
/* set the current color to white */
SetColor(f,VT_Black);

/* set the current position pointer to the upper left corner of rect. */
SetPosition(f,r->x,r->y);

/* drawm the border of the rectangle */
PaintRectangleBorder(f,r->w,r->h);

/* set the current position pointer to the upper left corner of the
   second rectangle                                                 */
SetPosition(f,r->x+4,r->y+4);

/* set the current color to white */
SetColor(f,VT_White);

/* paint inside the second rectangle to white */
PaintRectangleInterior(f,r->w-8,r->h-8);

/* set the current color to black */
SetColor(f,VT_Black);

/* set the current position pointer to the upper left corner of the
   second rectangle                                                 */
SetPosition(f,r->x+4,r->y+4);

/* draw the border of the second rectangle */
PaintRectangleBorder(f,r->w-8,r->h-8);

r ->genbool = TRUE;

/* set the current position pointer in to the middle of the rectangle */
SetPosition(f,((r->x+4)+(r->w-8)/2),((r->y+4)+(r->h-8)/2));

/* center the string relative to the current position pointer */
SetJustification(f,VT_CENTER);

/* draw the name of the object inside the rectangle */
PaintString(f,VT_STREND,r->name);
/* if object has a disjunctive relation .then draw a small circle
   under the upper border of the rectangle                        */
if (r->circbool == TRUE) {
    SetPosition(f,(r->x+r->w/2),r->y+3);
```

```
        PaintCircleBorder(f.4):
    } /* end if */
    boolcirc = FALSE:
} /* end function */


int UseableCoordinates()
{
    int option;
    struct wstate save:
    int fd:

    /* if the mouse is currently in the schema design area... */
    if ((input.u.mouse.x > DRAW_BOX_X)              &&
        (input.u.mouse.x < (DRAW_BOX_X+DRAW_BOX_W))  &&
        (input.u.mouse.y > DRAW_BOX_Y)              &&
        (input.u.mouse.y < (DRAW_BOX_Y+DRAW_BOX_H))) {
        return(1):
    } /* end if */
    /* if the mouse currently inside one of the type boxes... */
    else if ((input.u.mouse.x > TYPE_BOX_X)         &&
        (input.u.mouse.x < (TYPE_BOX_X+TYPE_BOX_W))  &&
        (input.u.mouse.y >  TYPE_BOX_Y)             &&
        (input.u.mouse.y < (TYPE_BOX_Y+TYPE_BOX_H))) {
        if (touchbool == TRUE)
            InvertRegion(f,(TYPE_BOX_X + blackobj * (2* BOX_WIDTH)),
                        TYPE_BOX_Y,(2*BOX_WIDTH),TYPE_BOX_H):
        touchbool = TRUE;

        /* find out in which type box the mouse currently put */
        current_type = ((input.u.mouse.x-TYPE_BOX_X)/(2*BOX_WIDTH)+4);

        /* invert the box where currently the mouse is put */
        if (current_type == 4)
            blackobj = 0;
        else if (current_type == 5)
            blackobj = 1;
        else if (current_type == 6)
            blackobj = 2;
        else if (current_type == 7)
            blackobj = 3;
        InvertRegion(f,(TYPE_BOX_X + blackobj * (2* BOX_WIDTH)),
```

```
                TYPE_BOX_Y,(2*BOX_WIDTH),TYPE_BOX_H);

      /* if the "OBJECT" box is selected... */
      if (((input.u.mouse.x-TYPE_BOX_X)/(2*BOX_WIDTH)) == 0) {

         /* present object_entry menu and get the name of the object and
            the relation names of the object                          */
         PresentMenu(&object_entry,answer);

         /* then present the attribute_menu and get the attribute names */
         PresentMenu(&attribute_menu.attr_answer);
      } /* end if */
      else if (((input.u.mouse.x-TYPE_BOX_X)/(2*BOX_WIDTH)) == 1) {
         PresentMenu(&genobj_menu.answer);
      } /* end else if */
   } /* end else if */
   /* if the circle box is touched... */
   else if ((input.u.mouse.x > CIRC_BOX_X)        &&
      (input.u.mouse.x < (CIRC_BOX_X+CIRC_BOX_W))  &&
      (input.u.mouse.y >  CIRC_BOX_Y)             &&
      (input.u.mouse.y < (CIRC_BOX_Y+CIRC_BOX_H))) {
      boolcirc = TRUE;
   } /* end else if */
   /* if one of the mode boxes is touched invert the region and change the
      mode                                                          */
   else if ((input.u.mouse.x > MODE_BOX_X)        &&
      (input.u.mouse.x < (MODE_BOX_X+MODE_BOX_W)) &&
      (input.u.mouse.y >  MODE_BOX_Y)             &&
      (input.u.mouse.y < (MODE_BOX_Y+MODE_BOX_H))) {
      InvertRegion(f,(MODE_BOX_X + current_mode * (2* BOX_WIDTH)).
                  MODE_BOX_Y,(2*BOX_WIDTH),MODE_BOX_H);
      current_mode = (input.u.mouse.x-MODE_BOX_X)/(2*BOX_WIDTH);
      InvertRegion(f,(MODE_BOX_X + current_mode * (2* BOX_WIDTH)).
                  MODE_BOX_Y,(2*BOX_WIDTH),MODE_BOX_H);
   } /* end else if */
   return(0);
} /* end function */



int IdentifyBox()
{
   short index;
```

```
   /* find out which record of the array is selected either to move or delete
      and return the record number of the rectangle. */
   for (index = npoints;index>=0;index--) {
      if ((input.u.mouse.x >= dlist[index].x-BOX_LINE)                    &&
          (input.u.mouse.x <= (dlist[index].x+dlist[index].w+2*BOX_LINE)) &&
          (input.u.mouse.y >= dlist[index].y-BOX_LINE)                    &&
          (input.u.mouse.y <= (dlist[index].y+dlist[index].h+2*BOX_LINE)))
              break;
   } /* end for */
   return(index);
} /* end function */


MoveBox()
{
   short box,x,y,w,h,i;

   /*if the mouse inside a rectangle.then get the ID number of the rectangle */
   if ((box = IdentifyBox()) >= 0) {
      x = dlist[box].x - BOX_LINE;
      y = dlist[box].y - BOX_LINE;
      w = dlist[box].w + (BOX_LINE*2);
      h = dlist[box].h + (BOX_LINE*2);

      /* track the movement of the mouse with the box until a mouse button is
         released.                                                        */
      TrackFixedBox(f.
                  &dlist[box].x,&dlist[box].y,dlist[box].w,dlist[box].h,
                  DRAW_BOX_X,DRAW_BOX_Y,DRAW_BOX_W,
                  DRAW_BOX_H,BOX_LINE);
      dlist[npoints+1] = dlist[box];

      for (i=box;i<=npoints;i++) {
         dlist[i] = dlist[i+1];
      } /* end for */

      RefreshBoxes(x,y,w,h);

      if (dlist[i].genbool == FALSE)
         DrawRectangle(&dlist[i]);

      else if (dlist[i].genbool == TRUE)
```

```
            DrawRectangle2(&dlist[i]);
    } /* end if */
    else {
        /* if the mouse is not in a rectangle.then draw a status bar near the
           bottom of the screen  */
        DisplayStatus(f."not a box");
    } /* end else */

} /* end function */



DeleteBox()
{
    short box,x,y,w,h,i;

    /* get the ID number of the box */
    if ((box = IdentifyBox()) >= 0) {
        x = dlist[box].x - BOX_LINE;
        y = dlist[box].y - BOX_LINE;
        w = dlist[box].w + (BOX_LINE*2);
        h = dlist[box].h + (BOX_LINE*2);

        /* delete the box */
        for (i=box;i<npoints;i++) {
            dlist[i] = dlist[i+1];
        }
        /* decrease the number of the boxes */
        npoints--;

        /* draw the new array of boxes */
        RefreshBoxes(x,y,w,h);
    } /* end if */
    else {
        /* if the mouse is not in a rectangle.then draw a status bar near the
           bottom of the screen */
        DisplayStatus(f."not a box");
    } /* end else */
} /* end function */



DrawBox()
```

```
{
   /* increase the number of the rectangles in the array */
   npoints++;

   /* put the x position of the mouse to the x coordinate of the rectangle */
   dlist[npoints].x = input.u.mouse.x;

   /* put the y position of the mouse to the y coordinate of the rectangle */
   dlist[npoints].y = input.u.mouse.y;
   dlist[npoints].w = 0;
   dlist[npoints].h = 0;

   /* put the object name.relation names and subobject names received from the
      user into the last added element of the rectangle */
   copy(objname.dlist[npoints].name);
   copy(ans1.dlist[npoints].rel1);
   copy(ans2.dlist[npoints].rel2);
   copy(ans3.dlist[npoints].rel3);
   copy(ans4.dlist[npoints].subobj1);
   copy(ans5.dlist[npoints].subobj2);
   copy(ans6.dlist[npoints].subobj3);

   /* clear the arrays which is used to get information from user */
   copy(dummy.objname);
   copy(dummy.ans1);
   copy(dummy.ans2);
   copy(dummy.ans3);
   copy(dummy.ans4);
   copy(dummy.ans5);
   copy(dummy.ans6);

   /* if the right most button is clicked.the width and height of the
      last element becomes the same as the element one before */
   if (repeat != 0) {
      if (npoints > 0) {
         dlist[npoints].w = dlist[npoints-1].w;
         dlist[npoints].h = dlist[npoints-1].h;
      } /* end if */
   } /* end if */
   /* track the movement of the mouse with the lower right corner of the
      rectangle until a mouse button is released and return the width and
      height of the rectangle */
   else {
```

```
        TrackRubberBox(1,dlist[npoints].x,dlist[npoints].y,
                       &dlist[npoints].w,&dlist[npoints].h,
                       DRAW_BOX_X,DRAW_BOX_Y,DRAW_BOX_W,
                       DRAW_BOX_H,BOX_LINE);
    } /* end else */
    /* if the movement of the mouse is to the left relative to the x position
       of the rectangle... */
    if (dlist[npoints].w < 0) {
        dlist[npoints].w = -dlist[npoints].w;
        dlist[npoints].x -= dlist[npoints].w;
    } /* end if */
    /* if the movement of the mouse is up relative to the y position
       of the rectangle... */
    if (dlist[npoints].h < 0) {
        dlist[npoints].h = -dlist[npoints].h;
        dlist[npoints].y -= dlist[npoints].h;
    } /* end if */
    switch (current_type) {
    case OBJECT_TYPE:
        if (boolcirc == TRUE)
            dlist[npoints].circbool = TRUE;
        else
            dlist[npoints].circbool = FALSE;
        DrawRectangle(&dlist[npoints]);
        break;
    case GENOBJ_TYPE:
        if (boolcirc == TRUE)
            dlist[npoints].circbool = TRUE;
        else
            dlist[npoints].circbool = FALSE;
        DrawRectangle2(&dlist[npoints]);
        break;
    } /* end switch */
} /* end function */


DrawLines()
{
    short i,a;
    int   width,height;

    /* check the relations between objects if there is a relation then
       draw a straight line between them */
```

```
    for (i=0:i  =npoints:i++) {
        for (a=0:a<=npoints:a++) {
            SetColor(f.VT_Black):
            if ((strcmp(dlist[i].rel1.dlist[a].name) == 0) ||
                (strcmp(dlist[i].rel2.dlist[a].name) == 0) ||
                (strcmp(dlist[i].rel3.dlist[a].name) == 0))   {
                Draw(&dlist[i],&dlist[a]):
            } /* end if */
        } /* end for */
    } /* end for */
} /* end function */


DrawDashedLines()
{
    short i.a:
    int   width.height:

    /* check if there is a relation between subobjects or between an object
       and subobject.if there is a relation.then draw a straight line between
       them.( it should actually be dashed line but since there is no library
       function to draw dashed line. a straight line is used instead  */
    for (i=0:i<=npoints:i++) {
        for (a=1:a<=npoints:a++) {
            SetColor(f.VT_Black):
            if (dlist[a].genbool == TRUE) {
                if ((strcmp(dlist[i].rel1.dlist[a].subobj1) == 0)  ||
                    (strcmp(dlist[i].rel2.dlist[a].subobj1) == 0)  ||
                    (strcmp(dlist[i].rel3.dlist[a].subobj1) == 0))    {
                    Draw(&dlist[i],&dlist[a]):
                } /* end if */
                if ((strcmp(dlist[i].rel1.dlist[a].subobj2) == 0)   ||
                    (strcmp(dlist[i].rel2.dlist[a].subobj2) == 0)   ||
                    (strcmp(dlist[i].rel3.dlist[a].subobj2) == 0))     {
                    Draw(&dlist[i],&dlist[a]);
                } /* end if */
                if ((strcmp(dlist[i].rel1.dlist[a].subobj3) == 0)   ||
                    (strcmp(dlist[i].rel2.dlist[a].subobj3) == 0)   ||
                    (strcmp(dlist[i].rel3.dlist[a].subobj3) == 0))     {
                    Draw(&dlist[i],&dlist[a]);
                } /* end if */
            } /* end if */
```

```
        if (dlist[i].genbool == TRUE) {
            if ((strcmp(dlist[i].subobj1.dlist[a].name) == 0)  ||
                (strcmp(dlist[i].subobj2.dlist[a].name) == 0)  ||
                (strcmp(dlist[i].subobj3.dlist[a].name) == 0))    {
                Draw(&dlist[i],&dlist[a]);
            } /* end if */
        } /* end if */
    } /* end for */
  } /* end for */
} /* end function */


Draw(r,s)
struct rectangle *r,*s;
{
   int width,height;

   /* take two rectangles that there is a relation between them and draw
     a line depending on the positoins of the rectangles          */

   if ((r->y+r->h) < s->y) {
      SetPosition(f,(r->x+r->w/2),(r->y+r->h));
      width = (s->x+s->w/2)-(r->x+r->w/2);
      height= s->y-(r->y+r->h);
      PaintLine(f,width,height);
   } /* end if */
   else if ((s->y+s->h)<r->y) {
      SetPosition(f,(s->x+s->w/2),(s->y+s->h));
      width = (r->x+r->w/2)-(s->x+s->w/2);
      height= r->y-(s->y+s->h);
      PaintLine(f,width,height);
   } /* end else if */
   else if ((r->y < s->y) && (r->x < s->x) && ((r->y+r->h)>s->y)) {
      SetPosition(f,(r->x+r->w),(r->y+r->h/2));
      width = s->x-(r->x+r->w);
      height= (s->y+s->h/2)-(r->y+r->h/2);
      PaintLine(f,width,height);
   } /* end else if */
   else if ((r->y > s->y) && (r->x<s->x) && (r->y < (s->y+s->h))) {
      SetPosition(f,(r->x+r->w),(r->y+r->h/2));
      width = s->x-(r->x+r->w);
      height= (s->y+s->h/2)-(r->y+r->h/2);
```

```
        PaintLine(f.width.height);
    } /* end else if */
    else if ((r->y < s->y) && (r->x>s->x) && ((r->y+r->h)>s->y)) {
        SetPosition(f,(s->x+s->w),(s->y+s->h/2));
        width =  r->x-(s->x+s->w);
        height= (r->y+r->h/2)-(s->y+s->h/2);
        PaintLine(f.width.height);
    } /* end else if */
    else if ((r->y > s->y) && (r->x>s->x) && (r->y < (s->y+s->h))) {
        SetPosition(f,(s->x+s->w),(s->y+s->h/2));
        width =  r->x-(s->x+s->w);
        height= (r->y+r->h/2)-(s->y+s->h/2);
        PaintLine(f.width.height);
    } /* end else if */
} /* end function */



strcmp(s,t)
char s[],t[];
{
    int i;

    i=0;
    while (s[i] == t[i])
        if (s[i++] == ' ')
            return(0);
    return(s[i]-t[i]);
} /* end function */
```

BIBLIOGRAPHY

Wu, C. T. **A New Graphics user Interface for Accessing Database.** Naval Postgraduate School. Department of Computer Science. Monterey. CA 93943.

Ullman, Jeffrey D.. **Principles of Database systems.** Computer Science Press, Inc.. Rockville, Maryland 20850, 1982.

Kroenke. D., **Database processing.** Science Research Associates, Inc.. Chicago, 1983.

Herot. Christopher F.. "Spatial Management of Data" **ACM Transactions on Database Systems.** December 1980. Volume 5. No. 4. Pages 493-514.

Daniel, B. and Hull. R.. **SNAP: A Graphics–based Schema Manager.** Computer Science Department, University of Southern California. Los Angeles, CA, 1986.

Stonebraker. M.. **TIMBER: A Sophisticated Relation Browser.** Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1982.

Wong, Harry K. T. and Kuo. I., **GUIDE: Graphical User Interface for database Exploration,** Lawrence Berkeley Lab.. University of California, Berkeley CA, 1982.

Braegger. Richard P. and Dudler A. and Rebsamen. J. and Zehnder C. A.. **Gambit: An Interactive database Design Tool for Data Structures. Integrity Constraints and Transactions.** Eidgenoessische Techische Hochschule(ETH), Institut fur Informatik, Zurich. Switzerland. 1984.

Rowe, L. A., **Fill-in-the-Form Programming.** Computer Science Division, EECS Department, University of California, Berkeley, CA, 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Computer Technology Curricular Officer, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
5.	C. T. Wu, Code 53Wq Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
6.	1STLT Alparslan Horasan Enerji Evleri, Imam Htp Ls. Kar. B. Blok Kat:4 Daire:28 Aydin / TURKEY	4
7.	Hv. K. K. Ligi Kutuphanesi Bakanliklar/Ankara TURKEY	2
8.	Hv. Egitim K. Ligi Kutuphanesi Guzelyali/Izmir TURKEY	1
9.	Hava Harp Okulu Kutuphanesi Yesilyurt/Istanbul TURKEY	1

219240

Thesis
H765
c.1

Horasan
Implementation Graphical Language for Accessing Database.